Janika Ranta K96384
Eero Peltonen 151179055
Teemu Lehtonen 151956838

OHJELMOINTI 3 DOCUMENTATION

Ohjelmointi 3
COMP.CS.140

Faculty of Information Technology and Communication Sciences
November 2023

Contents

Contents 1
Diagrams 2
Structure 2
Responsibilities of the key classes 2
  Backend 2
  Frontend 2
Functionality 3
  APIs 3
  Basic features 3
  Extra features 3
Teamwork 5
  Eero 5
  Janika 5
  Teemu 5
User manual 5
  Run application 5
  Run tests 6
Missing features and known bugs 6

Diagrams

The diagram of the program structure can be found in our git repository's Documentation folder. The diagram is made with UMLet 15.1 tool.

Structure

Project follows standard Maven conventions for directory structure and package organization in a JavaFX application. It has separate packages for controllers, models, utilities, and GUI-related components along with appropriate resources like FXML files and icons used in the application.

Summary of structure:

Controllers: Handle application flow and interaction logic.
Database Handling: Contains classes responsible for managing database-related operations.
Models: Define data structures and entities used within the application.
Utilities: Hosts various utility classes, such as API handling, file operations, and formatting.
Weather Application Components: Includes GUI-related classes, FXML files, and utilities for visualization.

Responsibilities of the key classes

The project design pattern is basic MVC (Model-View-Controller). View is implemented using FXML - technology. It makes the use of scene builders possible. FXML view uses GUI controllers to handle rendering, events and fetching data from backend. View makes the call to the controller which calls weather model. Weather model uses API class to make the request to the OpenWeatherMap API.

Backend

Backend's key classes are WeatherModel, API and RestClient. WeatherModel makes the call to API function. API class returns the response body as a string. WeatherModel class builds the WeatherDTO model from the response body. The model is built with Google's Gson class. This WeatherDTO model is then returned to WeatherController which is the controller between frontend and backend. WeatherController returns the model to GUI controller which then renders the results on the screen.

The structure of the API class is pretty simple. It implements iAPI interface which defines the functions for API to call. API uses RequestParameters and OpenWeatherRequest to build the url and headers to fetch the details from OpenWeatherMap API. OpenWeatherRequest class and RequestParameters class and other small classes are just utils classes to build the request. The user of the API class does not have to know how the request is constructed, so creating new API requests is easy.

The actual HTTP request is sent to external API in RestClient class. RestClient class has the getResponse method which handles the logic to send the HTTP request. RestClient then returns the HTTP request back to API then returns the body of the request to the WeatherModel, the functionality of which was described above.

Frontend

Frontend in this context is the graphical user interface and its controllers. We didn't use the provided base for graphical user interface and decided to create our own. Views were created with a view

specific fxml-file as a base layout and the layout was populated with methods. This gave us a preview on how the application will look and we could define visually how the UI will look when its dimensions are changed.

There is a controller for each separate view “Weatherapp_guiController” for the main view and “Search_guiController” for the view that is responsible for searching, favorites and search history. These controllers fill the data to the view, update the data on the views and react to user inputs.

VisualizationUtilities: main responsibility is to populate containers in the layouts with elements such as images and texts that matched the weather condition.

WeatherIconHandler: main responsibility is reading images from the resources folder.

Functionality

APIs

The application uses only two external APIs. The main API is OpenWeatherMap. Its responsibility is to return weather details for the application.

The other API is GeoNames. It returns the specific longitude and latitude details for cities. This API helped with implementing the search functionality of the application.

Basic features

The application has numerous features. The basic features that are implemented in the assignment instructions are listed below.

- The application shows the current weather and forecast for the next few days.
- Other weather icons than OpenWeatherMap are used.
- Interfaces are used in the code.
- Version history is visible in Gitlab.
- The favorites can be saved.
- Current location and favorites are saved on a disk (db.json file).
- Errors are handled when writing files.
- Unit tests have been implemented.

Extra features

- All classes and functions are documented with JavaDoc.

  We decided to comment all the classes because it helps using the code base. For example, if there has been a while since using the code, it is easy to get back on track if everything is clearly commented and documented.

- Location search history

  Location history is saved. We save every location which weather is shown in the weather view. We have a member called locationHistory in the db.json file. The file also contains a lastSearch

object the purpose of which is to save the last searched location. If the last search is not found, we use Tampere as the default location.

- Support of both imperial and metric systems

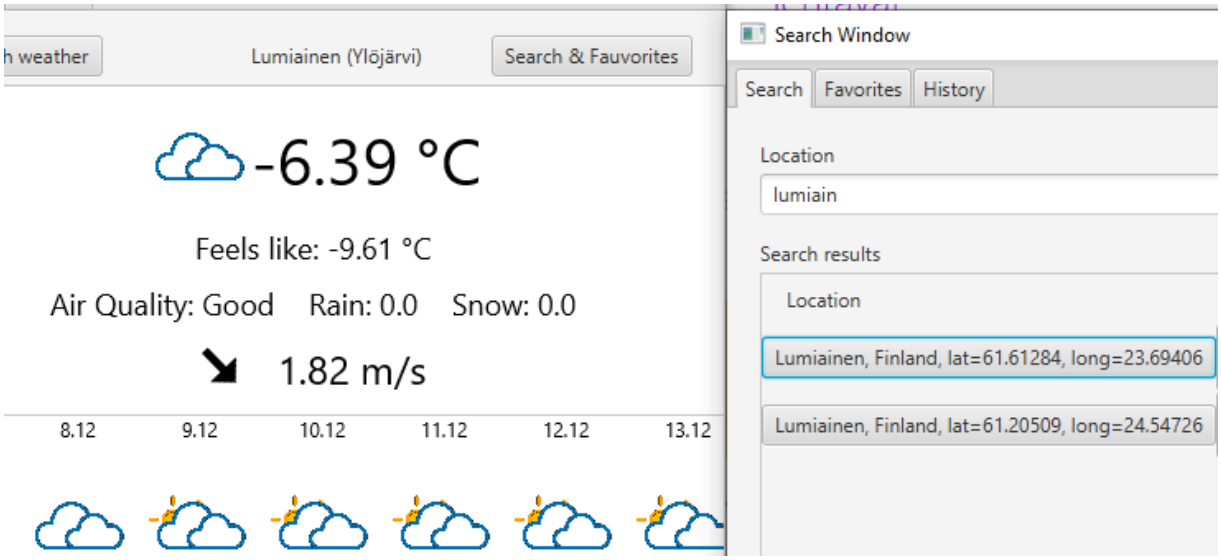
Our application supports both imperial and metric measuring systems. In the settings page we have an option where the user choice can be modified. Based on this user choice the correct values are fetched from API. The option is stored in the db.json file.

- Unit tests for UI have been implemented with TestFX

In addition to basic unit tests, we have decided to implement some unit tests for the UI. These tests ensure that the required buttons and details can be found from UI. These tests ensure the basic functionality of the UI. Tests also have own database which gets deleted after tests.

- Location search enhanced with GeoNames API (another API service)

The user can search small towns, villages and lakes with a partial name or name of the location. Search results contains name, country, and longitude- and latitude-coordinates. Those coordinates will then be used to find the closest matching weather station. Most smaller places don’t have their own weather stations, see the images below.



Teamwork

Our teamwork worked well. All the tasks were divided quite evenly. We used Telegram to communicate with each other. Version Control improved the teamwork because we agreed to use issues for different tasks. From Gitlab we could see what each individual is working on.

Our agreed workflow with version control was to push commits on separate branches and merge branches to main. This seemed to work really well as not many merge conflict issues appeared.

Collaboration felt professional and we didn't have miscommunication issues. Our team was very self-driven and team members didn't need much guidance from others.

Eero

My main task was the graphical user interface. I created the layouts for main- and search-views and most of the methods to fill them. I also added the GeoNames API for an enhanced location search functionality.

Janika

My main focus was the unit testing of the entire program. I focused on testing the core functionalities and did some small tweaks to them along the way. I applied the TestFX testing for the graphical user interface. I also made some small functions for the backend.

Teemu

My main task was to implement backend and API requests. I also did some small functionalities to UI. I wrote most of this document.

User manual

Run application

1. git clone git@course-gitlab.tuni.fi:compcs140-fall2023/group3191.git

2. cd group3191/WeatherApp
3. touch .env
4. Add following to .env file

API_KEY=<secret>

GEONAMES_USERNAME=group3191

USE_PRO_API=true # Add this if you have pro license to OpenWeatherMap API.

5. Mvn clean javafx:run

Run tests

1. Ensure you are in group3191/WeatherApp directory
2. mvn clean test

Missing features and known bugs

- We didn't implement any map for weather. This could have been a nice extra feature.
- Sometimes one GUI test fails. We think that the reason for this is a delay with API response. The test moves on but because we haven't got any response from API, elements have not yet been rendered on the screen and we run into an error. This only happens sometimes (10%).